



LIBRARY
OF THE
UNIVERSITY
OF ILLINOIS

510.84
I26r
no. 355-360
cop. 2



The person charging this material is responsible for its return on or before the **Latest Date** stamped below.

Theft, mutilation, and underlining of books are reasons for disciplinary action and may result in dismissal from the University.

UNIVERSITY OF ILLINOIS LIBRARY AT URBANA-CHAMPAIGN

NOV 1 1971
OCT 13 Rec'd

L161—O-1096

2262
no. 356

Math

Report No. 356

ICL: A CONTROL LANGUAGE FOR ILLIAC IV

by

Denise Christine Pavis

NOV 14 1969

October 13, 1969



DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN · URBANA, ILLINOIS

THE LIBRARY OF THE
UNIVERSITY OF ILLINOIS
CHAMPAIGN, ILLINOIS



Report No. 356

ICL: A CONTROL LANGUAGE FOR ILLIAC IV

by

Denise Christine Pavis

October 13, 1969

Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801

*This work was supported in part by the Advanced Research Projects Agency as administered by the Rome Air Development Center under Contract No. US AF 30(602)4144 and submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer Science, October, 1969.



Digitized by the Internet Archive
in 2013

<http://archive.org/details/iclcontrollangua356pavi>

ABSTRACT

This paper describes ICL, Illiac Control Language and the job parser which interprets the ICL. ICL is a free format, ALGOL-like language which allows the user to specify tasks to be processed in serial or parallel and provides for the conditional execution of tasks dependent upon the results of previously processed tasks. The job parser builds a tree which represents the ordering of tasks and dossiers, which contains in tabular form all information about a task and its associated files necessary to process the task.

ACKNOWLEDGMENT

The author would like to express her sincere appreciation to the members of the operating systems group and Mr. Peter Alsberg in particular for their invaluable advice and criticism during the design of ICL.

Mrs. Patricia Douglas must be thanked for her excellent job of typing the manuscript.

Finally, the author would like to express her gratitude to the Department of Computer Science and the ILLIAC IV project for its financial support and for the use of its facilities and the aid of its staff.

TABLE OF CONTENTS

	Page
1. INTRODUCTION	1
2. SYNTAX	2
2.1 Task	2
2.1.1 Introduction	2
2.1.2 Procedure Declaration	2
2.1.3 Procedure Body	3
2.1.4 Execute Card	3
2.1.5 I4 File Card	4
2.1.6 B6500 File Card	9
2.1.7 Examples of Procedures	9
2.2 Logic	10
2.2.1 Logic Card	10
2.2.2 Conditional Logic Statement	11
2.2.3 STOP	12
2.2.4 STEP	12
2.2.5 Switch Set Statement	12
2.2.6 Example of Logic Card	13
2.3 An ICL Program	13
2.3.1 Control Program	13
2.3.2 Library	15
3. IMPLEMENTATION: THE JOB PARSER	16
3.1 Introduction	16
3.2 The Tree Builder	16
3.3 The Doer	22

	Page
LIST OF REFERENCES	25
APPENDIX: ICL SYNTAX	26

LIST OF FIGURES

Figure	Page
1. Logic Tree	14
2. B6500 Disk File Name Block for MY/FIRST/PROGRAMLIB	21
3. Disk Blocks	23

LIST OF TABLES

Table	Page
1. THE ILLIAC DOSSIER	19
2. THE ILLIAC FILE DOSSIER	20
3. B6500 DISK FILE NAME BLOCK	21

1. INTRODUCTION

The Illiac Control Language, ICL, is the user's interface to ILLIAC IV. The job parser interprets the ICL and builds a tree which represents the entire job. Each node represents a task to be performed on the ILLIAC IV or the B6500 and points to a dossier which describes the task and the data files required by the task. Several tasks may be specified to be executed in parallel, since each node has a pointer to the next parallel task and a pointer to the next sequential task associated with it.

Several design criteria strongly influenced ICL. First, a user familiar with B5500 - B6500 control language should not have to learn an entirely new language. Second, the language should be free format. Third, the language should reflect the parallel capabilities of ILLIAC IV^{*} by allowing the user to specify tasks to be done in parallel. Fourth, the language should allow for conditional execution of tasks depending upon the results of completed tasks. Fifth, the job parser, the ICL interpreter, should be capable of interacting with the on-line user, reporting to him the status of his job and any errors in his ICL code. Sixth, as much advantage as possible should be taken of the B6500 MCP.

Chapter 2 describes the syntax of ICL. Chapter 3 describes the job parser and gives the format of the tables and dossiers built by the job parser.

* of a four-quadrant configuration.

2. SYNTAX

The syntax of ICL is specified in TBNF [1]. All identifiers in ICL have a maximum length of 17 characters and are represented by the symbol <*I> in the TBNF productions. The symbol <*N> represents an integer. The complete syntax is given in the Appendix; the following description is intended as an introduction to ICL and as a user's manual for the first edition.

2.1 Task

2.1.1 Introduction

An ICL task is defined to be a single program to be executed on the B6500 or the ILLIAC IV or any series of operations which can be specified by a legal sequence of B6500 control cards. Syntactically, a task or tasks may be declared as a procedure^{*} to be invoked by the appearance of a procedure call.^{**}

2.1.2 Procedure Declaration

<procedure declaration> ::= PROCEDURE <*I> <parameter list>? <insert>? ;

<procedure body>

<parameter list> ::= (LIST [<*I> <default>?] SEPARATOR,)

<default> ::= "=" <file identifier> [2]

The parameter list allows the same procedure to be used for many different purposes by changing the calling parameters. The default option specifies

^{*} Procedure declarations will not be permitted in the first edition, but keep reading as one will have to know what a procedure body is.

^{**} See 2.1.6.

that if the matching calling parameter does not occur then the parameter will be set to the default.

<insert> ::= IN <B6500 disk file name>

The insert option allows the user to automatically place the procedure into the library file^{*} given by the B6500 disk file name.

2.1.3 Procedure Body

<procedure body> ::= <execute card> | BEGIN <step block> END

<step block> ::= [<execute card> [<I4 file card> * | <B6500 file card> *]]* |
 \leq any legal B6500 control card sequence with the "?" replaced by
 "###" \geq

If more than one execution is specified by a procedure, the tasks represented are processed sequentially. A B6500 control card sequence is transferred to a zip file after replacing the "###" by "?" and is handed to the B6500 MCP [2] by the B6500 scheduler [3].

2.1.4 Execute Card

<execute card> ::= EXECUTE <switchset>? <program name> ON <machine name>

<time limit>? ;

<switchset> ::= <switch> :=

<switch> ::= SW< *N>

Switches SW1 through SW255 are global variables (type integer) available to the user to store the results of programs. If the program is a system

^{*} Library files are disk files containing ICL procedures. See 2.3.2.

program (e.g. the TRANQUIL compiler), the switch will be set by the system program to represent the relative success or failure of the program. If the program is a user program there will be a facility for passing an integer between 1 and $2^{24}-1$ to the operating system which will then set the switch to this value. Switches may be used in if statements^{*} to specify conditional execution of tasks.

<program name> ::= <B6500 disk file name>

<machine name> ::= [I4|ILLIAC IV] <number of quadrants>?|B6500

<number of quadrants> ::= ([1|2|3|4])^{**}

<time limit> ::= FOR <*N>

The time limit is given in terms of an integral number of seconds.

EXAMPLES: EXECUTE SW2 := A/B ON I4 ;

EXECUTE C/D ON B6500 FOR 5 ;

2.1.5 I4 File Card

<I4 file card> ::= FILE <*I> = [<initialization part> (<disk map part>); |
IMMEDIATE \leq data \geq ## ENDDATA ;]

The immediate form of the ILLIAC file card specifies that the file follows in line and is terminated by ## ENDDATA where "##" appears in columns one and two.

<initialization part> ::= [(<segments per record>? <motion part>)]?

<segments per record> ::= <*N>

^{*} See 2.2.2.

^{**} for four-quadrant configuration.

$\langle \text{motion part} \rangle ::= [\text{MOVE ON } \underline{\text{LIST}} [\langle \text{B6500 disk file name} \rangle \mid \# \langle *I \rangle] \\ \underline{\text{SEPARATOR}} \&]? [\text{MOVE OFF } [\langle \text{B6500 disk file name} \rangle \mid \\ \# \langle *I \rangle]]?$

The initialization part specifies the number of ILLIAC IV disk segments per record (default of one), where the file comes from and where it is to be placed when the task is completed. Several B6500 disk files may be concatenated to make one ILLIAC file at initialization time. In order to be saved a file must be moved off the ILLIAC disk.

$\langle \text{disk map part} \rangle ::= \langle \text{prefix} \rangle (\langle \text{map element list} \rangle \mid \langle \text{map element list} \rangle \mid \\ \text{SIZE OF } [\langle \text{B6500 disk file name} \rangle \mid \# \langle *I \rangle])$

The SIZE OF form of the disk map part implies that the user does not want to map his data onto the disk in any special manner and knows only that his file is the same size as some file on the B6500 disk.

$\langle \text{prefix} \rangle ::= \langle *N \rangle \mid [\&]? \langle \text{location} \rangle \mid \langle \text{location list} \rangle$
 $\langle \text{location} \rangle ::= \text{EU } \langle \text{euno} \rangle \mid \text{SU } \langle \text{suno} \rangle \mid \text{TRK } \langle \text{trkno} \rangle \mid \text{SEG } \langle \text{phase} \rangle$
 $\langle \text{location list} \rangle ::= (\underline{\text{LIST}} \langle \text{location} \rangle \underline{\text{SEPARATOR}} ,)$
 $\langle \text{euno} \rangle^* ::= 0 \mid 1$
 $\langle \text{suno} \rangle^* ::= 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 \mid 10 \mid 11$
 $\langle \text{trkno} \rangle^* ::= \leq \text{integer between 0 and 47 inclusive} \geq$
 $\langle \text{phase} \rangle ::= \leq \text{integer between 0 and 1199 inclusive} \geq$

The disk map part of the ILLIAC file card allows the user to map his data onto the disk in the manner which affords him the most efficiency. A prefix may indicate that the specification which follows is to be repeated one or

* These numbers are valid for the first system. They will be changed later.

more times; is to be placed on a particular electronics unit, storage unit, or track; or is to be phased relative to a particular segment number. Electronics units, storage units, tracks, and segments can be expressed in absolute or relative terms. The presence of "@" before a prefix indicates that the prefix is absolute, e.g. @EU1, @SU5 implies that space for the specification following is to be reserved on the EU and SU which are respectively numbered 1 and 5 by the hardware. Otherwise a prefix is relative, e.g. SU5 implies that the following specification is to be placed on any SU and, once chosen, that SU will be known as SU5 in case any later specifications require placement on the same SU.

EXAMPLES OF PREFIXES:

2 Make 2 copies of the following specification.

(@SU0, EU1) Space for the following specification is to be reserved on absolute storage unit 0 and relative electronic unit 1.

@SEG15 For the following specification relative segment 0 is to be set to absolute segment 15.

<map element list> ::= LIST <map element> SEPARATOR ,

<map element> ::= <*N> <qualifier> : <*N> <size qualifier> |
 <*N> <size qualifier> - <*N> <size qualifier> |
 <*N> <qualifier> : <repetitive element> |
 <*N> C <size qualifier> | <*N> <T qual> |
 <prefix> (<map element list>)

<Tqual> ::= [T | R | S] ≤ any alphameric string ≥ | < >

<size qualifier> ::= [R | S] ≤ any alphameric string ≥ | < >

<qualifier> ::= [D | M | R | S] ≤ any alphameric string ≥ | < >

<repetitive element> ::= (<*N> @ <*N> <qualifier> : <*N> <size qualifier>)

There are several forms of the map element; this variety allows the user to think of the disk in terms of degrees, milliseconds, segments or records. The user requests disk space in units of segments, records or tracks. (E.g. 200, 400S, 2TRK.) If there is no qualifier the request is assumed to be a request for records. He may ask for a contiguous portion of the disk by appending a "C" to his request (e.g. 200C, 400C SEG). He may ask that a request be phased with respect to a relative or absolute zero by preceding his request with a phase specification in terms of degrees, milliseconds, segments, or records. A phased request is by implication a contiguous request. (E.g. 10D: 100, 5MILLISEC: 100SEGMENTS.) Checking is done on the first letter of the qualifier; any blank free alphameric sequence is allowed to follow; hence D is equivalent to DEGREES is equivalent to DEG, etc.

The conversion between degrees and segments is 360° per 1200 segments. Hence 3° corresponds to 10 segments. Any phase request in terms of degrees which is not a multiple of three will be handled as if it were the next higher multiple of three (e.g. a request for 10DEG is equivalent to a request for 12DEG). The disk has a 40 millisecond rotational latency and hence one millisecond corresponds to 30 segments.

The repetitive element requests a given block of disk space to be repeated at a given interval for a specified number of times. For example, 5 @ 10MS: 7S reserves five blocks of seven segments where the first segments of any two consecutive blocks are separated by 10 milliseconds.

EXAMPLES:

10 DEG: 10	Starting at degree 10 (converted to segment 40) space for 10 records is reserved.
500C	Space for 500 contiguous records is reserved.
100S - 500S	Starting at segment 100, 500 segments are reserved.

2(@EU0(10D: 10, 100C), 50CS)

On absolute electronic unit 0, starting at degree 10, space for 10 records is reserved, space for an additional 100 contiguous records is reserved on absolute electronic unit 0. 50 contiguous segments are reserved anywhere on the disk. Two copies of the entire specification are made.

An entire I4 file card looks like:

FILE C = (2 MOVE ON A/B & A1/B1)(@ SEG5 (@EU0 (100: 500, 200: 500), @ EU1 (100: 500, 200: 500)), 1000C)

and conveys the following information:

- 1) The file is blocked two segments per record.
- 2) The file is to be initialized from B6500 disk file A/B concatenated with B6500 disk file A1/B1.
- 3) The file is not to be saved.
- 4) On absolute electronic unit 0, relative segment 0 is to be set to absolute segment 5. Space for 500 records (1000 segments) starting at relative segment 200 (absolute segment 205, relative record 100) and for 500 more records starting at relative segment 400 (absolute segment 405, relative record 200) is to be reserved on electronic unit 0.
- 5) The same request as 4 is to be repeated on absolute electronic unit 1.
- 6) 1000 additional contiguous records (2000 segments) are reserved; they may be anywhere on the disk.

2.1.6 B6500 File Card

```
<B6500 file card> ::= FILE <*I> = <file identifier>
```

```
<file identifier> ::= <B6500 disk file name> <options> | <immediate file> |
                        <unit file>
```

```
<options> ::= < to be specified >
```

<B6500 disk file name> ::= LIST <*I> SEPARATOR/

```
<unit file> ::= < to be specified >
```

```
<immediate file> ::= FILE <*D> = IMMEDIATE <data> ## ENDDATA
```

Unit files are not implemented in the first version. Immediate files have the same format and meaning as in I4 file card. Options will be identical to the options allowed by Burroughs 6500 MCP for their file cards (not as yet released) [2].

A B6500 file card which follows a B6500 execution request has the obvious meaning. However, a B6500 file card may appear after an ILLIAC IV execution request. In this case, the file referred to is assumed to be a data file for use by the job partner.

2.1.7 Examples of Procedures

```
PROCEDURE STEP1 (X = X1/Y1);
```

BEGIN

EXECUTE A/B ON ILLIAC IV FOR 5;

```
FILE C = (MOVE ON #X)((@SEGO, EU1, SU0)(0: 500, 1000: 100),
          10M: 50, 100)
```

FILE D = (SIZE OF M/N/Q);

END;

The #<*I> construct permits quick recognition of the parameter as it appears in the text. If in the procedure call the parameter X is not specified,

then file C will be initialized from file X1/Y1. Note that the parameters are not positional and may therefore occur in any order. Procedure STEP1 may be invoked by the following calls:

```
STEP1 (X = Q/R)
```

```
STEP1
```

```
PROCEDURE STEP2
```

```
BEGIN
```

```
EXECUTE SW5 := A/B ON B6500;
```

```
FILE M = IMMEDIATE;
```

```
    {data}
```

```
    ## ENDDATA;
```

```
END;
```

```
PROCEDURE STEP3
```

```
BEGIN
```

```
    ## COMPILE MY/PROG XALGOL SYNTAX;
```

```
    ## XALGOL FILE CARD = SOURCE/JOB1 SERIAL;
```

```
END;
```

2.2 Logic

2.2.1 Logic Card

```
<logic card> ::= BEGIN <logic body> END* | <procedure call>
```

```
<logic body> ::= LIST <parallel statement> SEPARATOR [NEXT | ;]
```

```
<parallel statement> ::= LIST <step> SEPARATOR [& | ALSO] | <set switch statement>
```

* A special terminal symbol following the "END" will most likely be added to the syntax later.


```

<set switch statement> ::= <switch> := [<switch> | <*N> | [MAX MIN]
                                (LIST <switch> SEPARATOR ,)]
<step> ::= <procedure call>* | <procedure body> | <file identifier>* |
          (<parallel statement>) | <conditional logic statement> | STOP
<procedure call> ::= <*I> <calling parameter list>?
<calling parameter list> ::= (LIST [<*I> = <file identifier>] SEPARATOR ,)
<conditional logic statement> ::= IF <boolean> THEN (<parallel statement>)
                                ELSE (<parallel statement>)
<boolean> ::= <switch> [< | > | = | NEQ | LEQ | GEQ> [<*N> | <switch>]

```

The logic card allows the user to specify the logical sequence in which his tasks are to be processed. Tasks separated by "ALSO" or "&" will be processed in parallel. The appearance of "NEXT" or ";" implies that all previous tasks must be completed before the next task is initiated. "ALSO" or "&" takes precedence over "NEXT" or ";" but parentheses allow the user to specify any ordering since the expression within the parentheses is evaluated first. The conditional logic statement provides further control by allowing conditional execution of tasks.

2.2.2 Conditional Logic Statement

In the conditional logic statement the value of a switch is compared to another switch or to an integer between 1 and $2^{24} - 1$. If the result of the comparison is true the parallel statement following the "THEN" will be executed; if the result is false the parallel statement following the "ELSE" will be

* not implemented in first edition.

executed. The parallel statements following the "THEN" and "ELSE" must be enclosed in parentheses.*

EXAMPLES:

```
IF SW10 > 5 THEN (STOP) ELSE (A & B)
```

```
IF SW20 NEQ SW17 THEN (A) ELSE (IF SW6 LEQ 200 THEN (A) ELSE  
(C NEXT D) ALSO R)
```

2.2.3 STOP

The special reserved word "STOP" when executed stops all further processing and indicates that the user's job is to be terminated.

2.2.4 STEP

A step then can be a conditional logic statement, a "STOP", a procedure call, or a procedure, or it can be a file identifier. The file referred to by the file identifier must contain more ICL which is inserted at this point. When the ICL on this file is exhausted control will be returned to the original file.

2.2.5 Switch Set Statement

To avoid race conditions set switch statements can occur only after a ";" or "NEXT". Switches can be set from other switches, to any integer between 0 and $2^{24} - 1$ or the maximum or minimum of a list of switches (where the list has maximum length of six switches).

*Note: Go tos are not allowed since before any task can be sent off to be processed all tasks that precede it must be completed. Keeping track of all possible paths and the corresponding number of preceding tasks per path that must be completed would increase the complexity of the job parser by an estimated factor of 2.

EXAMPLES:

SW1 := SW5

SW1 := 257

SW17 := MAX (SW2, SW3, SW4, SW6)

2.2.6 Example of Logic Card

The job parser interprets the user's ICL creating a tree which represents the logical structure of the user's job. Each node of the tree is an encoding of a conditional logic statement including a true and a false pointer corresponding to the THEN and ELSE branches, or a switch setting statement which has only a sequential pointer, or a task to be executed which has a sequential and a parallel pointer, or a join which brings together the THEN and ELSE branches and which has a parallel and a sequential pointer, or a STOP which has no pointers.

```
A & IF SW2 > 10 THEN (C & (D NEXT F)) ELSE (G & H NEXT R) & Z NEXT SW5:= 200
NEXT Q;
```

The above ICL generates the symbolic tree of Figure 1 where S = sequential, P = parallel, T = true, F = false and an "X" in the pointer indicates that the pointer is null. Single letter identifiers are procedure calls.

2.3 An ICL Program

2.3.1 Control Program

```
<control program> ::= <accounting card>; [<library card>;] *
                    <procedure declaration> * <logic card>
<accounting card> ::= ≤ to be specified ≥
```

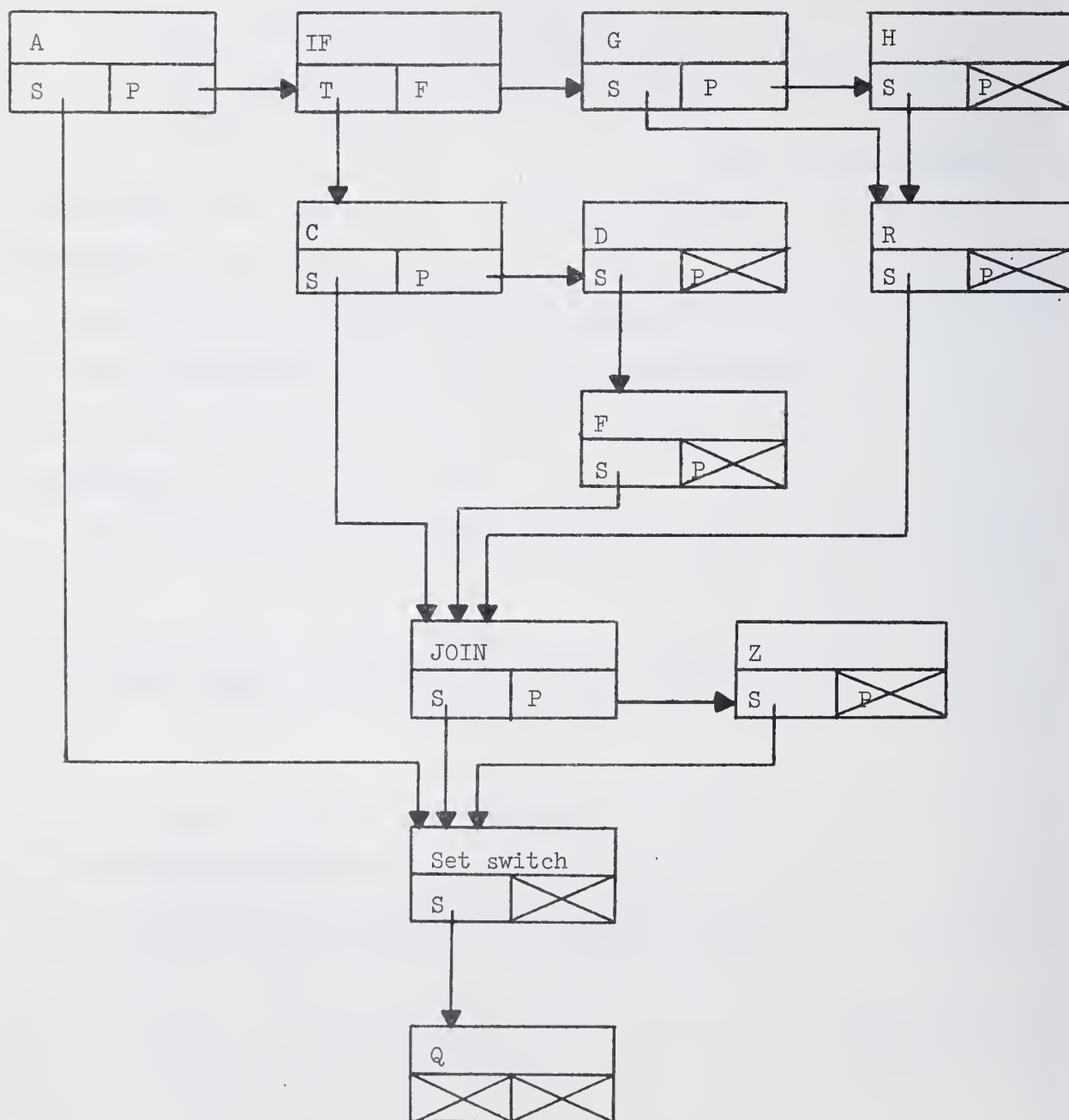


Figure 1. Logic Tree.

2.3.2 Library

<library card> ::= LIBRARY = LIST <B6500 disk file name> SEPARATOR ,

A library card gives a list of disk files or libraries which contain procedures. Any procedure called in logic card must be declared or be found in a declared library.

EXAMPLE:

LIBRARY = LIB1/NAME/PROCS, LIB2/PROCS

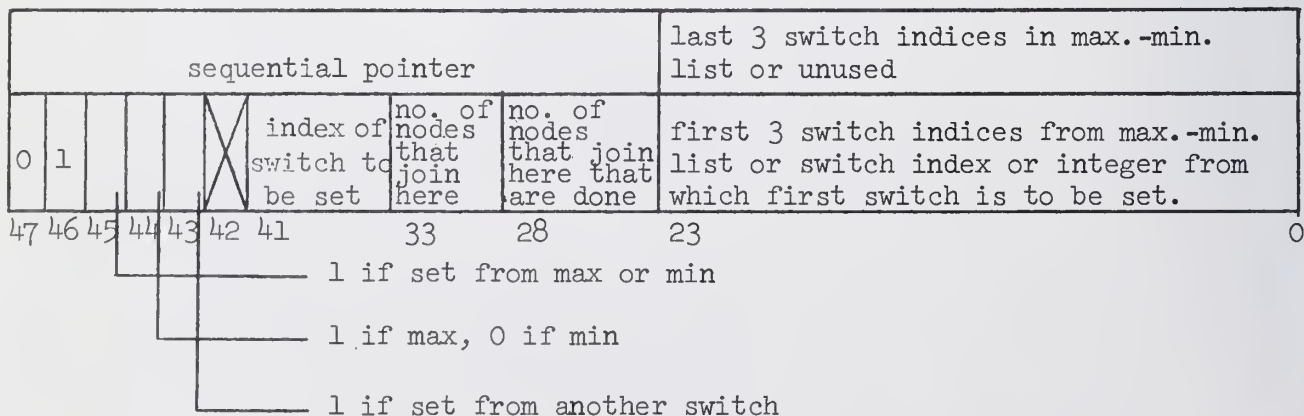
3. IMPLEMENTATION: THE JOB PARSER

3.1 Introduction

There is a job parser attached to each user. The job parser interprets the user's ICL, builds a tree which describes the logical structure of the user's tasks and dossiers or tables which describe the tasks, initiates tasks, and cleans up after the completion of a task. The job parser is really two modules; the first, the builder, builds the tree and the dossiers. The second, the doer, runs down the tree sending off ILLIAC tasks to the disk file allocator and B6500 tasks to the B6500 scheduler. The builder will be discussed first.

3.2 The Tree Builder

As mentioned in the syntax description section, each type of step in the logic card generates a different type of node in the tree. A set switch step generates this node:



EXAMPLE: SW10:= MAX (SW2, SW3, SW4, SW5)

not filled in yet								5		
0	1	1	1	0	X	10	not filled in yet	filled by doer	3	4
47	46	45	44	43	42	41	33	28	23	0

A conditional logic statement generates this block:


true pointer							false pointer		
0	0	switch index	compar- ison code		no. join here	no. nodes joining done	switch index or integer to be com- pared to		
47	46	45	37	34	33	28	23		0
				1 if compare to switch 0 if compare to integer					
				EQL 0 LSS 2 GTR 3 NEQ 1 LEQ 4 GEQ 5					

EXAMPLE:


IF SW10 > 200 THEN (A & B) ELSE (C) generates:

pointer to A							pointer to C		
0	0	10	3	0	not filled in yet	not filled in yet	200		
47	46	45	37	34	33	28	23		0

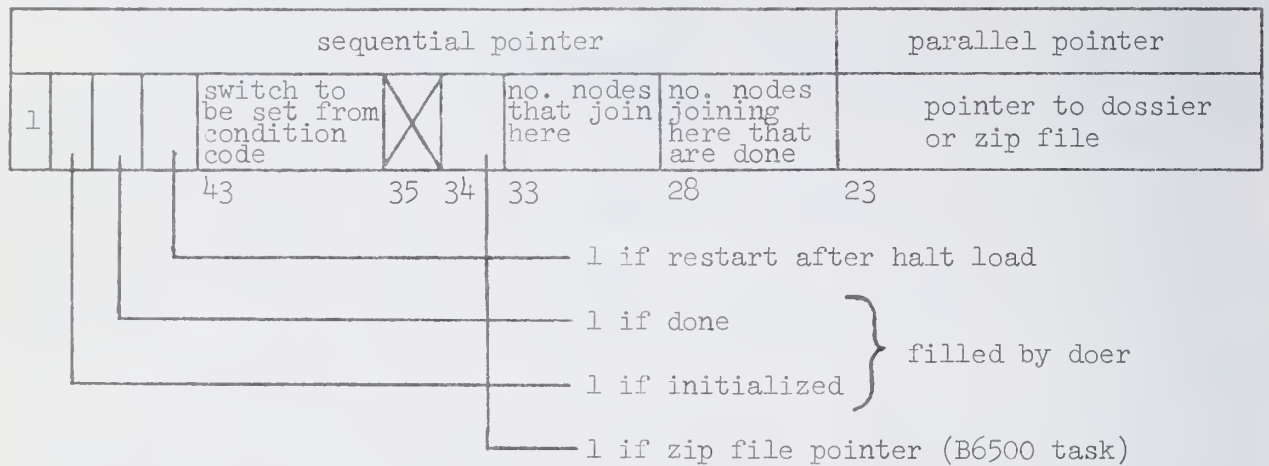
The join node that corresponds to each if has the following format:

sequential pointer										parallel pointer	
0	0	no. nodes joining here from false branch		0	0		no. nodes joining here from true branch	no. of nodes that are done	pointer to if that gener- ated the join		
47	46		40	39	38	37	34	33	28	23	0
0 if corresponding if is false, 1 if true (filled in by doer)											

EXAMPLE:

not filled in yet									not filled in yet	
0	0	1	filled by doer	0	0		2	filled by doer	pointer to if	
47	46	45	40	39	38	37	33	28	23	
									0	

A procedure call or a procedure body generates an execution node with the following format:



Procedure logic determines what type of step the ICL specifies. If it is a procedure call or procedure body, procedure EXECUTECARD is called to determine whether the task is an ILLIAC or B6500 task. If it is a B6500, a zip file is created and a pointer to the zip file is inserted in the corresponding node. If it is an ILLIAC task, dossiers describing the task are generated. The ILLIAC dossier has the format specified in Table 1.

The ILLIAC dossier points to the disk file requests. Each ILLIAC task generates at least one file request--that of the program file. Each additional ILLIAC IV file generates a new dossier, and all file dossiers are linked together. The format of the file dossiers is given in Table 2. They are generated by procedure I4FILE CARD which parses the initialization part and which calls PREFIX and MAPLIST to parse the disk map part.

B6500 file cards referring to job partner data files are handled by storing the card image in a disk block and pointing to it from the ILLIAC dossier.

Since the external B6500 disk file name may be arbitrarily long, special blocks are designed to contain these names. The pointer to the job

TABLE 1
THE ILLIAC DOSSIER

	<u>WORD</u>	<u>FIELD</u>	<u>FIELD CONTENTS</u>	
L	0:1	[47:24]	link to the next older lead dossier	(DISK ALLOC)
		[23:24]	link to the next younger lead dossier	(DISK ALLOC)
L	1:1		job ID number	(JOB PARSE)
LS	2:1		job step number	(JOB PARSE)
L	3:3		project ID	(JOB PARSE)
L	6:3		user ID	(JOB PARSE)
L	9:3		access code	(JOB PARSE)
FS	12:1	[47:24]	pointer to the next sequential step	(JOB PARSE)
		[23:24]	pointer to the next parallel step	(JOB PARSE)
S	13:1		step condition code	(EXEC MONIT)
FS	14:1	[47:24]	pointer to the file map table	(DISK ALLOC)
		[23:24]	pointer to the file name table	(DISK ALLOC)
L	15:1	[47:24]	pointer to the preprocess requests	(DISK ALLOC)
		[23:24]	pointer to the postprocess requests	(DISK ALLOC)
	16:1	[47:24]FS	pointer to the disk file requests	(JOB PARSE)
		[23:24]S	pointer to the job partner name	(JOB PARSE)
S	17:1		maximum time for this step element (NS)	(JOB PARSE)
L	18:1		maximum time for the whole job (MS)	(JOB PARSE)
	19:1	[47:8]L	category requested	(JOB PARSE)
		[39:8]L	category received	(DISK ALLOC)
		[31:8]FS	step level	(JOB PARSE)
		[23:8]FS	number of steps that join here	(JOB PARSE)
		[15:2]S	number of quadrants required	(JOB PARSE)
		[13:1]L	dossier built	(JOB PARSE)
		[12:1]L	request queued	(DISK ALLOC)
		[11:1]L	file space allocated	(DISK ALLOC)
		[10:1]L	data preprocessed	(D PROCESSR)
		[9:1]L	execution completed	(EXEC MONIT)
		[8:1]L	data postprocessed	(D PROCESSR)
		[7:1]L	file space freed	(DISK ALLOC)
L	20:1	[33:8]	year (69, 70, etc.):	} time of entry into system
		[25:4]	month:	
		[21:5]	day:	
		[16:17]	time (seconds):	
L	21:1		time of entry into disk file allocator	(DISK ALLOC)
S	22:1	[23:24]S	pointer to job partner data files	(JOB PARSE)
	23:7		reserved for future expansion	

TABLE 2
THE ILLIAC FILE DOSSIER

<u>WORD</u>	<u>FIELD</u>	<u>FIELD CONTENTS</u>
0:1	[47:24]	number of words used in this block
	[23:24]	pointer to next block
1:3		file name
4:3		unused
7:1	[47:24]	number of segments/record
	[23:24]	total number of segments in the file
8:1	[47:24]	pointer to preprocess requests
	[23:24]	pointer to postprocess requests

Starting at word 8 and continuing until the file is exhausted are two-word blocks specifying the segment relations. The file specification is terminated by a two-word block containing zeros.

9:1	[47:24]	segment number to start at
	[22:24]	number of segments, including above, involved in this specification
10:1	[43:1] = 1	if specific segment is required
	[42:1] = 1	if the space is contiguous
	[41:1] = 1	if the space is phased
	[40:1] = 1	if a virtual EU is required
	[39:1] = 1	if a virtual SU is required
	[38:1] = 1	if a virtual TRK is required
	[37:1] = 1	if a specific EU is required
	[36:1] = 1	if a specific SU is required
	[35:1] = 1	if a specific TRK is required
	[34:4]	EU number
	[30:8]	SU number
	[22:10]	TRK number
	[12:13]	phase (in terms of segments)

partner, pointer to preprocess and pointer to postprocess requests point to these kinds of blocks. The format is given in Table 3 and an example in Figure 2.

TABLE 3

B6500 DISK FILE NAME BLOCK

0:1 [47:24] number of words used in this block
 [23:24] pointer to next block

Starting at word 1 and continuing until the name is exhausted are three-word blocks. The n-th block contains the number of characters and the n-th identifier or the name.

1:1 [47:6] number of characters of identifier (for BCL input)
 [41:42] first 7 characters of identifier
 2:2 remainder of identifier


0	9							
1	2	M	Y					
2								
3								
4	5	F	I	R	S	T		
5								
6								
7	10	P	R	O	G	R	A	
8	L	I	B				M	
9								
10								
29								

Figure 2. B6500 DISK FILE NAME BLOCK FOR MY/FIRST/PROGRAMLIB.

The tree builder might be best understood by considering an example of a task and constructing the appropriate disk blocks.

```
EXECUTE SWLO := SYS/PARSER ON ILLIAC IV (1) WITH PARSER/PARD FOR 5;
FILE B = (8 MOVE ON A/B MOVE OFF C/D)((@EUL, @SUO, @TRKO)
        (0: 600 SEGMENTS), SUL(4(0: 10, LOMS: 20)),
        100C, 500, 200C)
```

generates the disk blocks of Figure 3.

3.3 The Doer

The doer runs in parallel^{*} with the builder. It begins at node 0 of the tree and runs down the tree sending ILLIAC IV tasks to the disk file allocator and B6500 tasks to the B6500 scheduler. It also checks for tasks that have been completed, freeing the disk blocks belonging to that task.

A task is sent off provided that all tasks which logically precede it have been completed. The doer works by chasing down the parallel pointers, sending off tasks, until a null or empty parallel pointer is reached. It will then back up and chase down sequential pointers. When it can send off no more tasks, it will check for completions and begin again.

The doer proceeds according to the following algorithm:

1. Number of nodes that join here is not equal to number of nodes that join here that are done--put node in WAITS stack and go to 2, else continue.
- 1.1 Stop node--quit.
- 1.2 If node--evaluate if, go to true or false node.
- 1.3 Set switch node--set the switch, go to next sequential node.
- 1.4 Join node--go to 1.5.2.

^{*} Assuming two processors on the B6500, it is started after the builder so that it will have some data to work with.

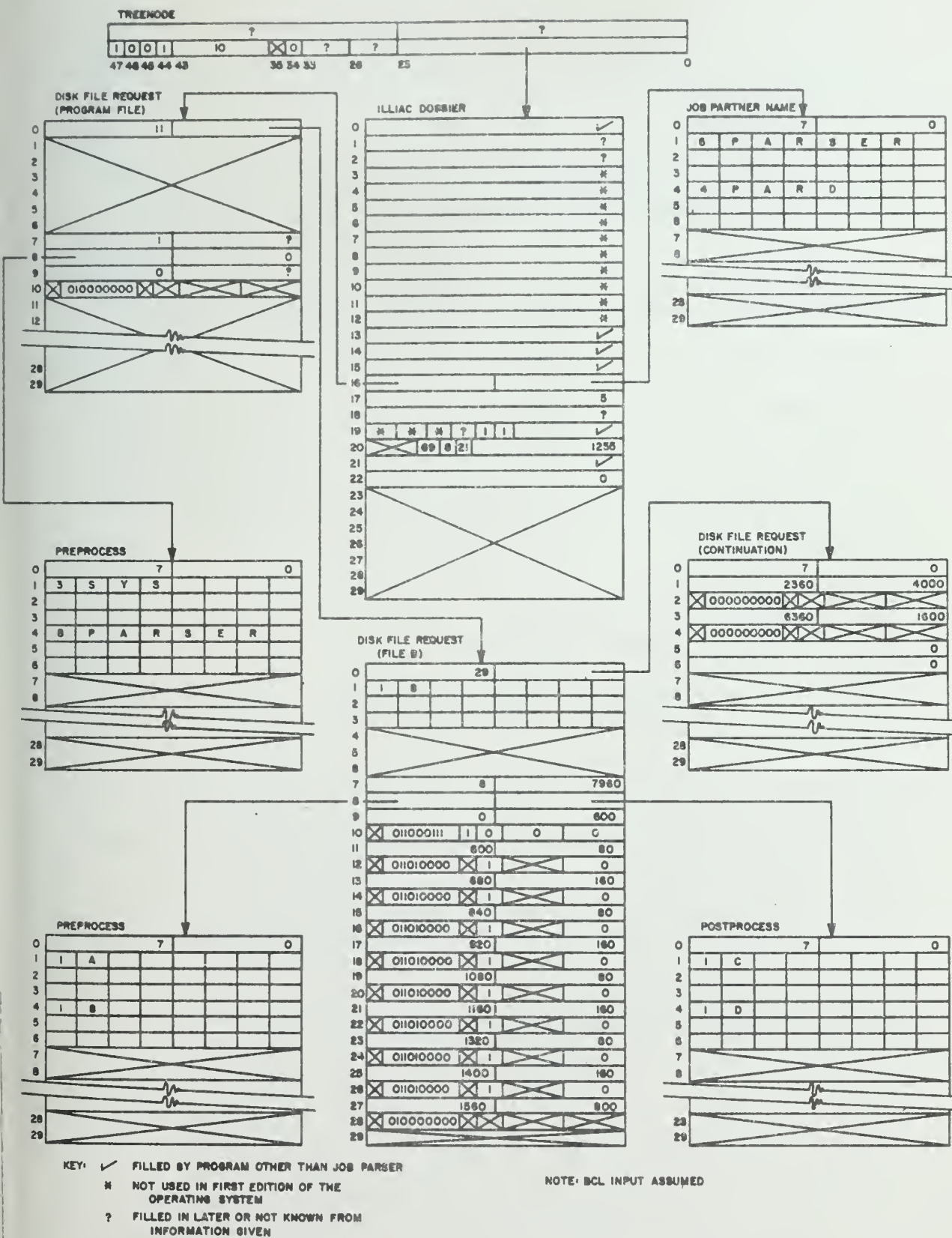


Figure 3. Disk Blocks.

1.5 Execution node

1.5.1 Send off task to disk file allocator or B6500 scheduler.

1.5.2 Both parallel and sequential pointers empty, put node in PAREMPTY and SEQEMPTY stacks, go to 2.

1.5.3 Sequential pointer empty, put node in SEQEMPTY stack; parallel pointer null, go to 2, else get node pointed to by parallel pointer and go to 1.

1.5.4 Parallel pointer empty, put node in PAREMPTY stack; sequential pointer null, go to 2, else get node pointed to by sequential pointer and go to 1.

1.5.5 Sequential pointer not null, parallel not null, put node in SEQSKIP stack if not there already, get node pointed to by parallel pointer and go to 1; parallel pointer null, get node pointed to by sequential pointer and go to 1.

1.5.6 Sequential pointer null, parallel pointer OK, get node pointed to by parallel pointer and go to 1.

1.5.7 Both null, quit.

2. Get here if nothing else can be sent off.

2.1 Something has been completed, set done bit, repeat.

2.2 No more completions.

2.2.1 Check WAITS if anything can be done, now get node number and go to 1.

2.2.2 Go through the SEQEMPTY stack to see if any sequential pointers have been filled in--if so, transfer node to SEQSKIP stack if not there already.

2.2.3 Go through PAREMPTY stack if any of the parallel pointers have been filled in--get node and go to 1.

2.2.4 Go through SEQSKIP stack--get first node and go to 1.

2.2.5 Go to 2.

LIST OF REFERENCES

- [1] Trout, H., "A BNF Like Language for the Description of Syntax Directed Compilers", Report No. 300, Department of Computer Science, University of Illinois at Urbana-Champaign (January 1969).
- [2] Burroughs Corporation, "Master Control Program Design Concepts for the B6500/B7500 Information Processing System", Systems Programming Department, Business Machines Group, Burroughs Corporation, Pasadena, California (June 1969).
- [3] Alsberg, P., and Mills, C., "The Structure of the ILLIAC IV Operating System", to be published by ACM, Second Symposium on Operating Systems Principles (October 1969).

APPENDIX

ICL SYNTAX

```

<CONTROL PROGRAM>::=<ACCOUNTING CARD> <LIBRARY CARD>*
    <PROCEDURE DECLARATION CARD>*<LOGIC CARD>
<LIBRARY CARD>::=LIBRARY=<FILE NAME LIST>;
<FILE NAME LIST>::=LIST<R6500 DISK FILE NAME> SEPARATOR ,
<PROCEDURE DECLARATION CARD>::=PROCEDURE<+I> <PARAMETER LIST>?
    <INSERT>?;<PROCEDURE BODY>;
<PARAMETER LIST>::=(LIST[<+I><DEFAULT>?])SEPARATOR ,)
<DEFAULT>::= <FILE IDENTIFIER>
<INSERT>::= IN <R6500 DISK FILE NAME>
<PROCEDURE BODY>::=<EXECUTE CARD>/BEGIN <STEP BLOCK> END
<EXECUTE CARD>::=EXECUTE <SWITCHSET>? <PROGRAM NAME> ON
    <MACHINE NAME> <TIME LIMIT>?;
<SWITCHSET>::=<SWITCH>;
<SWITCH>::=SW<+N>
<PROGRAM NAME>::=<FILE IDENTIFIER>
<MACHINE NAME>::=[I4/ILLIACIV/ILLIAC IV][(1/2/3/4)1?/B6500
<TIME LIMIT>::=FOR <+N>
<STEP BLOCK>::=[<EXECUTE CARD>[I4 FILE CARD>*/
    <R6500 FILE CARD>+]*]*/
    <ANY LEGAL R6500 CONTROL CARD SEQUENCE WITH ?
    REPLACED BY ##>
<LOGIC CARD>::=BEGIN <LOGIC BODY> END/<PROCEDURE CALL>
<LOGIC BODY>::=LIST <PARALLEL STATEMENT> SEPARATOR[;/NEXT]
<PARALLEL STATEMENT>::=LIST<STEP>SEPARATOR[&/ALSO]/
    <SET SWITCH STMT>
<SET SWITCH STMT>::=<SWITCH>:[<SWITCH>/<+I>/
    (MAX/MIN)(LIST<SWITCH>SEPARATOR,)]

```



```

<STEP> ::= <PROCEDURE CALL> / <PROCEDURE BODY> /
    <FILE IDENTIFIER> / ( <PARALLEL STATEMENT> ) /
    <CONDITIONAL LOGIC STATEMENT>
<PROCEDURE CALL> ::= <+I> <CALLING PARAMETER LIST> ?
<CALLING PARAMETER LIST> ::= ( LIST [ <+I> = <FILE IDENTIFIER> ]
    SEPARATOR , )
<CONDITIONAL LOGIC STATEMENT> ::= IF <BOOLEAN> THEN
    ( <PARALLEL STATEMENT> ) ELSE ( <PARALLEL STATEMENT> )
<BOOLEAN> ::= <SWITCH> [ </> ] = / <NEQ / LEQ / GEQ> [ <+N> ] / <SWITCH> ]
<14 FILE CARD> ::= FILE <+I> * [ <INITIALIZATION PART>
    ( <DISK MAP PART> ) ] / IMMEDIATE <DATA> ## ENDDATA
<DISK MAP PART> ::= <PREFIX> ( <MAP ELEMENT LIST> ) /
    <MAP ELEMENT LIST> / SIZE OF <R6500 DISK FILE NAME>
<PREFIX> ::= <+N> / [ <+> ] ? <LOCATION> / <LOCATION LIST>
<LOCATION> ::= E [ <EUNO> / SU <SUNO> / TRK <TRKNO> / SFG <PHASE> ]
<PHASE> ::= <ANY INTEGER BETWEEN 0 AND 11992>
<LOCATION LIST> ::= ( LIST <LOCATION> SEPARATOR , )
<EUNO> ::= 0 / 1
<SUNO> ::= 0 / 1 / 2 / 3 / 4 / 5 / 6 / 7 / 8 / 9 / 10 / 11
<TRKNO> ::= <ANY INTEGER BETWEEN 0 AND 47>
<MAP ELEMENT LIST> ::= LIST <MAP ELEMENT> SEPARATOR ,
<MAP ELEMENT> ::= <+N> <QUALIFIER> [ <+N> <SIZE QUALIFIER> /
    <+N> <SIZE QUALIFIER> - <+N> <SIZE QUALIFIER> /
    <+N> <QUALIFIER> [ <REPETITIVE FLT> /
    <+N> <SIZE QUALIFIER> /
    <+N> <TQUAL> /
    <PREFIX> ( <MAP ELEMENT LIST> )
<TQUAL> ::= [ R / S / T ] <ANY ALPHAMERIC SEQUENCE> / <>
<SIZE QUALIFIER> ::= [ R / S ] <ANY ALPHAMERIC SEQUENCE> / <>
<QUALIFIER> ::= [ D / E / F / S ] <ANY ALPHAMERIC SEQUENCE> / <>
<REPETITIVE FLT> ::= ( <+N> @ <+N> <QUALIFIER> [ <+N> ] )

```

```

<INITIALIZATION PART>::=(<SEGMENTS PER RECORD>?
    <MOTION PART>)/<>
<SEGMENTS PER RECORD>::=<+N>
<MOTION PART>::=MOVF ON(<SOURCE>)
    MOVF OFF([<B6500 DISK FILE NAME>/#<+I>])
<SOURCE>::=LIST[<B6500 DISK FILE NAME>/#<+I>] SEPARATOR &
<B6500 FILE CARD>::=FILE <+I>=<FILE IDENTIFIER>
<B6500 DISK FILE NAME>::=LIST<+I> SEPARATOR/
<UNIT FILE>::=<TO BE SPECIFIED>
<FILE IDENTIFIER>::=<B6500 DISK FILE NAME><OPTIONS>
    /<IMMEDIATE FILE>/<UNIT FILE>/#<+I>
<OPTIONS>::=<TO BE SPECIFIED>
<IMMEDIATE FILE>::=FILE<+I>=IMMEDIATE SDATA? ## FNDDATA
<ACCOUNTING CARD>::=<TO BE SPECIFIED>

```


UNCLASSIFIED

Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

ORIGINATING ACTIVITY (Corporate author)

Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801

2a. REPORT SECURITY CLASSIFICATION

UNCLASSIFIED

2b. GROUP

REPORT TITLE

ICL: A CONTROL LANGUAGE FOR ILLIAC IV

DESCRIPTIVE NOTES (Type of report and inclusive dates)

Research Report

AUTHOR(S) (First name, middle initial, last name)

Denise Christine Pavis

REPORT DATE

October 13, 1969

7a. TOTAL NO. OF PAGES

36

7b. NO. OF REFS

3

a. CONTRACT OR GRANT NO.

46-26-15-305

b. PROJECT NO.

US AF 30(602)4144

9a. ORIGINATOR'S REPORT NUMBER(S)

DCS Report No. 356

9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)

10. DISTRIBUTION STATEMENT

Qualified requesters may obtain copies from DCS.

11. SUPPLEMENTARY NOTES

NONE

12. SPONSORING MILITARY ACTIVITY

Rome Air Development Center
Griffiss Air Force Base
Rome, New York 13440

13. ABSTRACT

This paper describes ICL, Illiac Control Language and the job parser which interprets the ICL. ICL is a free format, ALGOL-like language which allows the user to specify tasks to be processed in serial or parallel and provides for the conditional execution of tasks dependent upon the results of previously processed tasks. The job parser builds a tree which represents the ordering of tasks and dossiers, which contains in tabular form all information about a task and its associated files necessary to process the task.

DD FORM 1473
1 NOV 65UNCLASSIFIED
Security Classification

UNCLASSIFIED

Security Classification

14.

KEY WORDS

LINK A

LINK B

LINK C

ROLE

WT

ROLE

WT

ROLE

WT

Illiac Control Language

Job parser

UNCLASSIFIED

Security Classification

UNIVERSITY OF ILLINOIS-URBANA

510.84 IL6R no. C002 no.355-360(1969

Report /



3 0112 088398810